Document

document 1 of 1 Order Document

INSPEC - 1969 to date (INZZ)

Accession number & update
5980412, C9809-7810C-031; 980729.

Title
Developing interactive educational engineering software for the World Wide Web with Java.

Author(s)
Reed-J-A; Afjeh-A-A.

Author affiliation
Dept of Mech, Ind & Manuf Eng, Toledo Univ, OH, USA.

Source
Computers-Education (UK), vol.30, no.3-4, p.183-94, April-May 1998. , Published: Elsevier.

CODEN
COMEDR.

ISSN
ISSN: 0360-1315, CCCC: 0360-1315/98/ ($19.00+0.00).

Availability
SICI: 0360-1315(199804/05)30:3/4L.183:DIEE; 1-T
Electronic Journal Document Number: S0360-1315(97)00062-6.

Publication year
1998.

Language
EN.

Publication type
J Journal Paper.

Treatment codes
P Practical.

Abstract

The World Wide Web has emerged as an effective mechanism for distributing educational material to students beyond the bounds of the classroom. The introduction of mobile code, such as Java applets, combined with the pervasiveness of the World Wide Web, present the potential to significantly enhance the development and distribution of educational software. This paper illustrates the design and implementation of a Java applet for use in propulsion engineering curricula. The Gas **Turbine Simulator** applet provides an interactive graphical environment which allows the rapid, efficient construction and analysis of arbitrary gas **turbine** systems. The simulation system couples a graphical user interface and a transient, space-averaged, aero-thermodynamic gas **turbine** analysis method, both entirely coded in the Java language. The combined package provides analytical, graphical and data management tools which allow the student to construct and control dynamic gas **turbine**

simulations by manipulating graphical objects on the computer display screen. The **simulator,** running as a Java applet, can be easily accessed from the World Wide Web and run from a variety of heterogeneous computer platforms, including PCs, Macintosh and UNIX machines, through the use of Java-enabled Web browsers. (28 refs).

Descriptors

aerodynamics; courseware; digital-simulation; engineering-education; engineering-graphics; gas-turbines; graphical-user-interfaces; information-dissemination; interactive-systems; Internet; mechanical-engineering-computing; microcomputer-applications; object-oriented-programming; propulsion; thermodynamics.

Keywords

interactive educational engineering software; World Wide Web; educational material distribution; mobile code; Java applets; propulsion engineering curricula; Gas **Turbine Simulator;** interactive graphical environment; graphical user interface; transient space averaged aerothermodynamic gas **turbine** analysis method; analytical tools; graphical tools; data management tools; dynamic gas **turbine** simulations; heterogeneous computer platforms; IBM PC compatible microcomputers; Apple Macintosh; UNIX machines; Java enabled Web browsers.

Classification codes

C7810C	(Computer-aided instruction).
C7210	(Information services and centres).
C6110J	(Object-oriented programming).
C6130B	(Graphics techniques).
C6180G	(Graphical user interfaces).
C7440	(Civil and mechanical engineering computing).
C7220	(Generation, dissemination, and use of information).

Copyright statement

save locally as: PDF document include search strategy

order

A JAVA-ENABLED INTERACTIVE GRAPHICAL GAS TURBINE PROPULSION SYSTEM SIMULATOR

John A. Reed*
Abdollah A. Afjeh†

The University of Toledo
Toledo, Ohio

Abstract

This paper describes a gas turbine simulation system which utilizes the newly developed Java™ language environment software system. The system provides an interactive graphical environment which allows the quick and efficient construction and analysis of arbitrary gas turbine propulsion systems. The simulation system couples a graphical user interface, developed using the Java Abstract Window Toolkit, and a transient, space-averaged, aero-thermodynamic gas turbine analysis method, both entirely coded in the Java language. The combined package provides analytical, graphical and data management tools which allow the user to construct and control engine simulations by manipulating graphical objects on the computer display screen. Distributed simulations, including parallel processing and distributed database access across the Internet and World-Wide Web (WWW), are made possible through services provided by the Java environment.

Introduction

Designing and developing new aircraft propulsion technologies is a time-consuming and costly process. The Numerical Propulsion System Simulation (NPSS) project, being developed at the at the NASA Lewis Research Center, aims to address these problems in part through the development of a software system providing detailed design and analysis capability of propulsion systems [1]. A key component of this system is the software simulation framework which supports the development, control and execution of the various analytical, database and visualization codes which are needed in a detailed propulsion system simulation.

For the past several years, we have been researching the development of such an environment. In previous work, a prototype aircraft propulsion simulation environment was developed using the Application Visualization System (AVS) [2, 3]. That system provided an interactive graphical programming environment with which to explore the concepts of object-based modelling and simulation, distributed and parallel processing over local and wide-area networks, component zooming, and scientific visualization [4, 5, 6, 7, 8]. While the objectives of the research were met with the use of AVS, the AVS system's primary design intent was not of aircraft propulsion simulation, and thus limited implementation of various simulation concepts.

To completely realize our goal of a sophisticated environment for propulsion system simulation, the framework needed to be re-implemented using a fully object oriented language [9]. In the AVS-based framework, the simulation model utilized object oriented concepts, but was incomplete since programming was done in Fortran and C. For example, multiple instantiation of the same class of object was provided by running each object in a separate UNIX process. This provided data abstraction and encapsulation, but meant that the work was limited to platforms offering multi-processing capability. Furthermore, message passing between objects was awkward and inefficient and there was no support for inheritance. Additional requirements of the simulation framework dictated that it be capable of distributed heterogeneous computing support, database management and access, ability to leverage existing legacy Fortran and C codes, and graphics capability for development of a graphical user interfaces. Several object-oriented languages (Smalltalk, C++, CLOS, etc.) available in late 1994 were evaluated, but each suffered from one or more deficiencies. Most had either vendor or platform specific implementations which placed limitations on the distribution of the simulation code. Support for distributed computing, database access and graphical libraries required third-party support which increased the complexity of the system. In early 1995, a new programming software, known as Java, became available. It held promise as both a programming language and run-time environment for our simulation framework.

The initial motivating factor for using Java was that it was a freely-available, fully object-oriented language. Java, however, provides additional advantages over other languages. First, Java utilizes an architecturally-neutral byte-code format which provides a high degree

* University Fellow, Student Member AIAA
† Associate Professor, Department of Mechanical Engineering, Member AIAA

of portability. That standard format has allowed all major computer operating systems vendors to quickly implement the Java VM on their architectures. Additionally, Java-enabled browsers, such as Netscape™, are capable of running Java programs. Java also provides built-in support for distributed computing, including such functions as remote object instantiation and migration, and access to the World-wide Web (WWW). SQL interfaces for accessing databases are also available and the language provides consistent interfaces for future integration of additional features.

In this paper, we describe the initial work in developing the simulation framework using Java. The next section presents a brief overview of the main characteristics of Java. The following section describes the Java Gas Turbine Simulation software system which represents a first-step in developing a Java-based framework for aircraft propulsion simulation. Finally, the last section presents a summary and lists the direction of future work.

Java

Java is a newly created programming language and environment developed by Sun Microsystems Inc. [10] Although originally designed for embedded logic control in electronic devices, Java has found success as a programming software for use on the Internet due to its capability as an ideal environment for the development of secure, distributed, network-based end-user applications capable of running on a variety of computer platforms. Java is both a programming language and supporting run-time environment.

The Language

Java, the language, is a general purpose object-oriented (OO) programming language offering OO capabilities such as data abstraction, encapsulation, polymorphism, and inheritance [9]. Although syntactically similar to the C++ programming language, Java removes many of the shortcomings of C++ which make C++ complex and confusing. In addition, Java adds many of the better object-oriented features available in other object-oriented languages (such as Smalltalk, Eiffel, Objective C) which are missing in C++. The most notable enhancement is the elimination of pointer-arithmetic, which does away with the possibility of overwriting and corrupting data; and similarly, the use of garbage collection, which allows the system to allocate and reclaim memory dynamically (i.e. without the programmer specifically calling these functions).

Run-time Environment

Java is an interpreted system: Java source code is compiled into byte-codes, rather than native machine code. This provides an architecturally neutral file format, which permits a compiled Java program to be transported to platforms of differing architecture. The Java program can then be run on any computer which implements the Java interpreter and run-time system. This interpreter and run-time system is known as the Java Virtual Machine (JVM). The use of byte-coding and implementation of the JVM allows a Java program to achieve a high degree of portability: a Java code compiled into byte-codes is portable to any machine which implements the JVM.

In addition, the Java run-time provides packages which support accessing network resources. The run-time system provides support for internet connections through Uniform Resource Locator (URL) methods, allowing a Java program to access files across the World-wide Web. Stream network connections are also available through the Socket mechanism supported by the run-time system.

Software programs written in Java are classified as either an Applet or an Application. Applets are typically small Java programs which are embedded in another application, such as an HTML document, and execute within a Java-enable browser. The browser implements the Java runtime interpreter as well as most graphical support for the applet. Applets are launched by embedding an applet HTML tag in a web page and then viewing that page with a Java-enabled web browser. When the page is accessed, the applet's byte codes are transferred across the net from the server to the client computer. On the client computer, the applet byte-code is then automatically loaded and run by the Java interpreter built into the web browser. Because they load from machines on the network, applets are restricted from implementing certain functions for security reasons. For example, applets may not read, write or modify files on the local (client) machine.

Java applications are usually larger programs developed for stand-alone operation. Applications are not run within a Java-enabled browser, and thus are not automatically loaded across the net and installed on the local computer; they must be installed manually in the traditional manner. Furthermore, a Java runtime system must be installed on the local computer in order to run the program. Because an application must be installed by the user, not downloaded automatically from a network, it is assumed to be secure. Thus, applications are not restricted by the same security issues as an applet and file access and modification are permitted.

Java Gas Turbine Simulation Software

The Java Gas Turbine Simulation software provides the user with the ability to graphically construct arbitrary gas turbine configurations, select and control steady-state and transient operation of the system, and view results in graphical form as the simulation is executing. The Java Gas Turbine Simulation software is entirely written in Java and can be run as either an Applet or an Application.

For full implementation as a simulation framework, the Java Gas Turbine Simulation software is run as an Application. This allows the necessary functionality, such as file access, which would otherwise be restricted due to the security features of an Applet. In some cases, it may be desirable to run the Java Gas Turbine Simulator as an Applet. This can be useful, for example, in teaching students how gas turbine systems operate, where making it accessible from Java-enabled browsers located on heterogeneous, networked computers provides an advantageous way to distribute the software [11].

The remainder of this section discusses the gas turbine engine analytical model and the graphical user interface (GUI) for the Java Gas Turbine Simulation software. The engine model and GUI are both entirely coded in the Java language. Additionally, the model and GUI are designed to be largely independent of one another so that modifications to either the model or GUI do not affect the other.

Analysis Mathematical Model

The mathematical model used to describe the operation of the gas turbine system in the current work is patterned after that presented in [12]. A complete description of the model can be found in [8]. Here, the gas turbine system is decomposed into its individual basic components: inlet, compressor, combustor, turbine, nozzle, bleed duct connecting duct, and connecting shaft. Intercomponent mixing volumes are used to connect two successive components as well as define temperature and pressure at component boundaries. Operation of each of the components is described by the equations of aero-thermodynamics which are space-averaged to provide a lumped-parameter model for each component. For dynamic (transient) gas turbine operation, the model includes the unsteady equations for fluid momentum in connecting ducts, inertia in rotating shafts, and mass and energy storage in intercomponent mixing volumes. Overall performance maps are used to provide accurate steady-state representations of compressor and turbine component operation. Variable geometry effects in the compressor are accounted for using baseline maps that correspond to nominally scheduled geometry. Variable geometry maps are then used to bias the baseline map outputs by functions of the actual geometry.

When the individual components in the gas turbine system are combined, each connecting-duct, connecting-shaft, and mixing volume contributes its respective unsteady equation to form a system of unsteady ordinary differential equations. This system may then be solved using standard numerical techniques such as Newton-Raphson, Runge-Kutta, etc.

Analysis Model Class Structure

The current analysis model class structure is based on the work described in [13]. In that work, the DIGTEM code and analysis model were decomposed using a component/control volume approach to create a fundamental object representation for a lumped-parameter model of a gas turbine engine. From that representation, a hierarchical class structure was fashioned which was then used to develop a LISP version of the DIGTEM code. This class hierarchy served as a starting point for the Java version of the gas turbine analysis class structure.

For the Java simulator class hierarchy, modifications were made to the LISP class structure to take advantage of lessons learned in our previous research as well as to account for differences in the LISP and Java languages. For example, the LISP model employed multiple inheritance in its class structure. Java does not support multiple inheritance, so the class structure was re-designed to a single inheritance hierarchy (see top of Figure 1).

Using this new class structure, a Java-enabled gas turbine analysis program was developed. The code allows modelling of steady and transient operation of the gas turbine through the application of the space-averaged, conservative, aero-thermodynamic equations. The object-oriented nature of the Java language allows the program to model almost any gas turbine engine by combining specified types of component objects in almost any order. Currently, the following component types are available: inlet, duct, combustor, compressor, turbine, shaft and nozzle. Several abstract classes, such as GasDynamicEntity, Source, Rotator, etc., are used to define characteristics and methods common to several inherited classes.

Although designed to be used with a graphical user interface, the engine analysis code may be run without the GUI using a user-written input file which defines the components in the model, their connections, and solver selection and control.

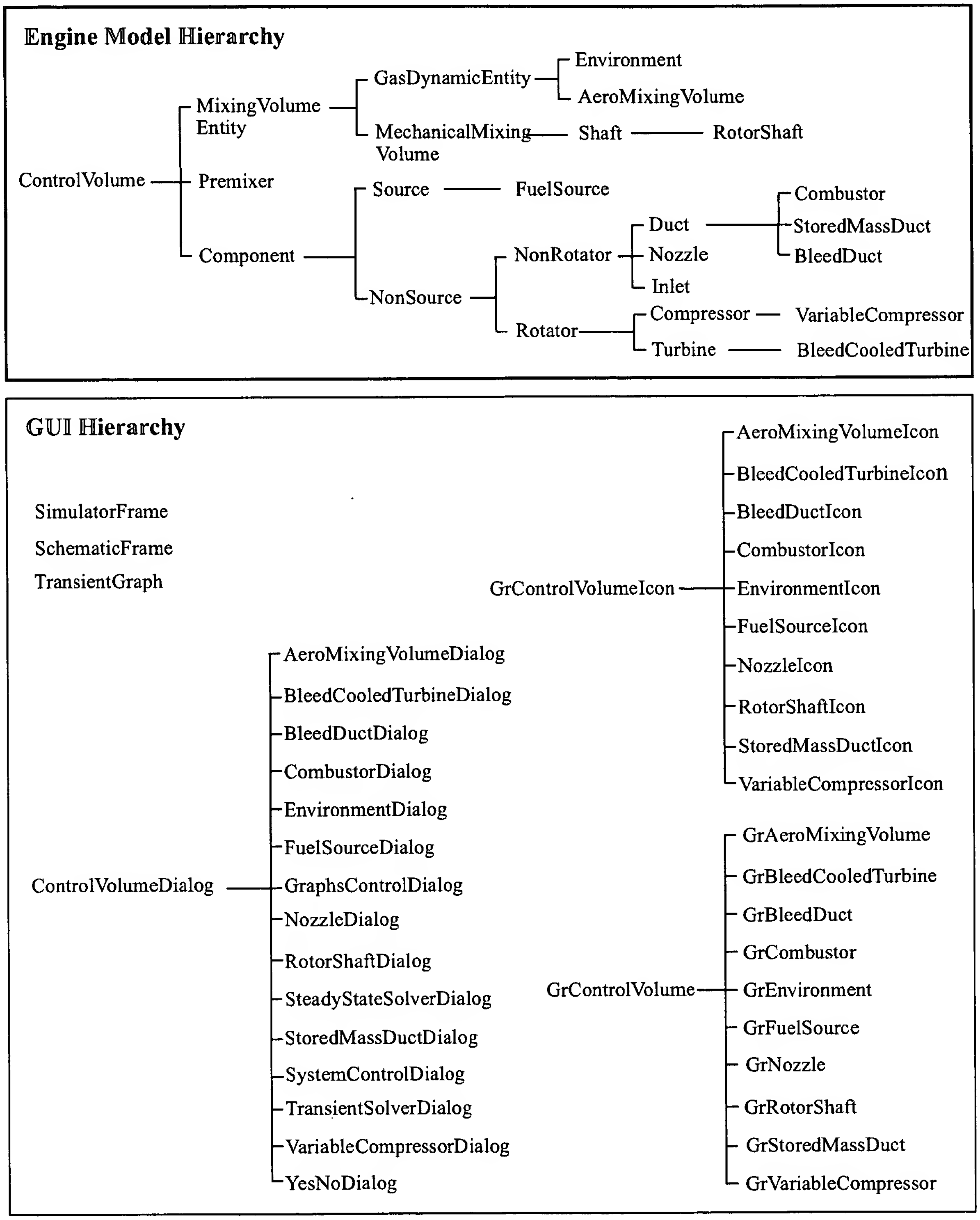

Figure 1: Java Gas Turbine Simulator Engine and GUI class hierarchies

Graphical User Interface and Class Structure

The graphical user interface (GUI) provides the user with interactive control of the gas turbine analysis model described above. The complete GUI is constructed entirely using the Java Abstract Windowing Toolkit, which provides the basic windowing components necessary to create graphical interfaces.

The Java Abstract Window Toolkit (AWT) provides a collection of platform-independent graphical components for building graphical applications in Java. The AWT, which is part of the Java runtime system, offers support for graphics operations, as well as supplying common graphical user-interface objects such as Buttons, Lists, TextFields, Choice boxes, etc. Platform independence is achieved through the use of *Peers* which are native GUI components manipulated by the AWT. Peers allow Java programs that use the AWT to retain the familiar look and feel of the host computer's native windowing system. This means that when a Java program is run on a Macintosh computer, the Buttons, Windows, List boxes, etc., all appear in the familiar Macintosh style. When the same program is run on a computer running Windows 95™, those Buttons, Windows and List objects will appear as familiar Window 95-style objects. The platform independence of the AWT, combined with the portability offered by the Java language make Java an ideal language and runtime system for use on networked, heterogeneous computer systems such as the World-wide Web.

The Java language has built-in support for multi-threading. Multi-threaded execution (also known as lightweight process) allow the program to create multiple paths of executing code. Using multi-threading can significantly improve the performance of an applications graphical-user interface (GUI).

The GUI is based on the Model-View-Controller (MVC) paradigm made popular in the Smalltalk-80 programming environment [14]. Using this implementation, the engine analysis code serves as the Model and is separated from the View, or graphical portions of the simulation code. Controllers, in the form of Dialogs, implement functions for accessing and updating the data in the Model. By implementing MVC and thus separating the Model and View, the data used in both the View and Model is described only once. Using MVC, the overall system possesses a high degree of modularity and thus is portable and extendible.

The GUI class structure consists mainly of six major classes which represent the main components of the GUI. These are shown in the bottom of Figure 1. The `SimulatorFrame`, `SchematicFrame`, and `TransientGraph` extend the AWT `Frame` class which provide basic windowing support. Classes `ControlVolumeDialog`, `GrControlVolumeIcon` and their subclasses define the engine component dialogs and icons, respectively. Class `GrControlVolume` and its subclasses are graphical representations of their respective engine component model objects defined in the engine model hierarchy.

Overview of Interface Components

The Java Gas Turbine Simulator user interface consists of 5 major windows. This section provides an overview of each of these windows, describing the functionality which they provide.

Main Window

Starting the Java Gas Turbine Simulator displays the *Main* window (see Figure 2). From there the user can access the various main windows of the simulation system: *Engine Schematic Layout*, *System Control Dialog*, *Graphing*, *Transcript*, or *Exit* the simulation system.

Engine Schematic Layout Window

A gas turbine simulation model is developed by building a schematic representation of the gas turbine in the Engine Schematic Layout window (see Figure 3). Individual engine components are represented graphically as Icons with each class of engine object (e.g., BleedDuct, VariableCompressor, etc.) having a uniquely shaped and colored graphical representation. color

The user selects engine components to add to the model from the Components menu located in the EngineSchematicLayout menu bar. The component's icon is displayed in the work area and a Dialog window for that engine object is also displayed. The user defines the operational characteristics for the component (i.e., the component name, design- and initial-operating point performance data, etc.) in the engine component's dialog window. For example, Figure 4 shows the dialog window for an AeroMixingVolume component. Similar dialogs, with appropriate design- and operating-point

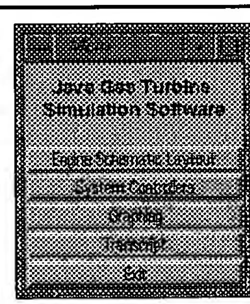

Figure 2: *Main* window

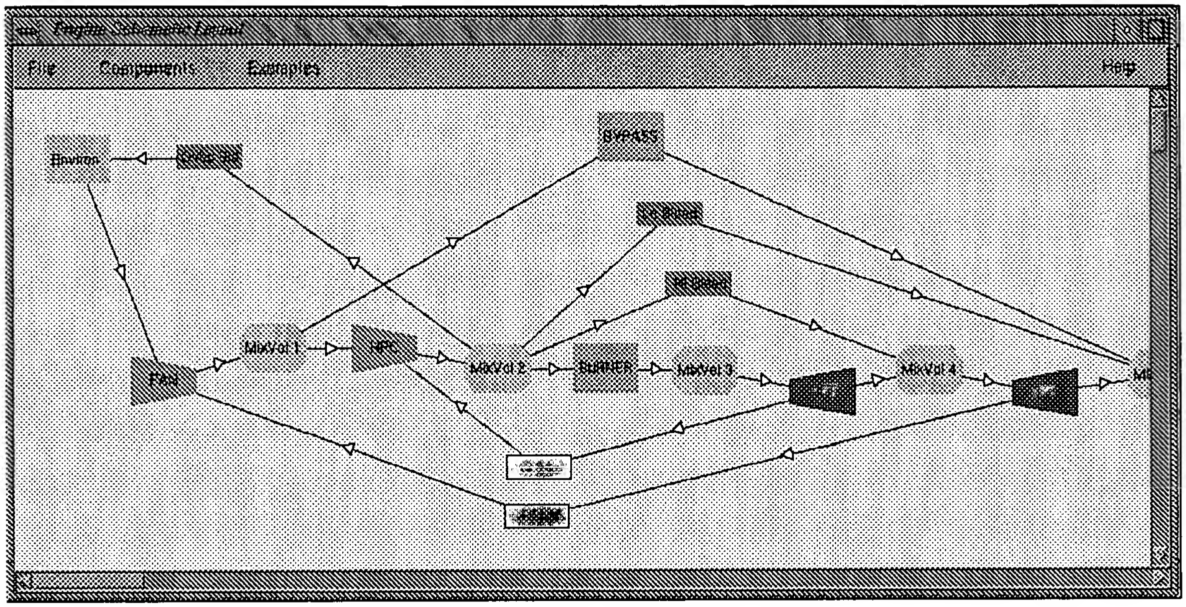

Figure 3: *Engine Schematic Layout* window

parameters, are used for each of the other engine components.

Additional engine components are added as needed to complete the engine model. Once all of the icons have been placed on the schematic work area, they may be dragged into place and interconnected to create the engine schematic diagram. In the diagram, the arrow-headed connecting lines represent both the directional flow path for fluid through the engine, and the structural connections along which mechanical energy is transmitted. Figure 3 shows the *Engine Schematic* 1
Layout window with a typical gas turbine engine model.

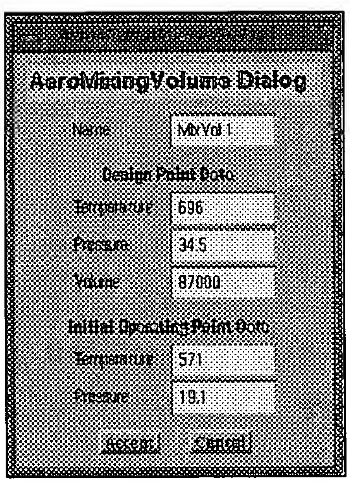

Figure 4: *AeroMixingVolumeDialog* window

System Control Dialog window

The *System Control Dialog* window (see Figure 5), which provides controls for the overall operation of the simulation, is accessed by depressing the System Controllers button in the *Main* window (see Figure 2). 2
The steady-state numerical solver used to balance the gas turbine equations at the initial operating point is selected from a list of available solvers. This list is displayed using a Choice component which displays the current list selection. In Figure 5, the Newton-Raphson method has been selected as the steady-state solver. Similarly, the transient solver may be selected from the list of available transient solvers. Here, the Improved Euler method has been selected.

The selected solver's parameters are edited by depressing the steady-state or transient Edit Configuration buttons. This displays a dialog with which the user can 3
define specific control values for the solver's operation. Similar dialogs are available for each of the available solvers, and selecting a different solver from the list will bring up different control parameters specific to the newly selected solver.

Below both the steady-state and transient solver selectors are the steady-state and transient *Progress* 4
Indicators. These indicators act as gauges which provide visual feedback to the user during the simulation indicating the progress of the steady-state

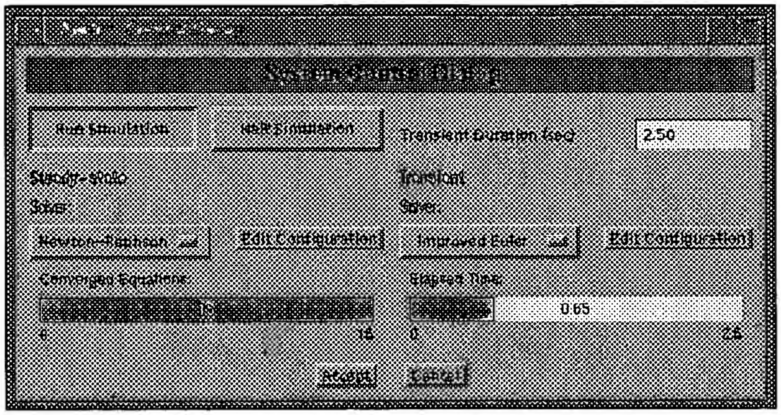

Figure 5: *System Control Dialog* window

balancing or transient integration processes. The steady-state *Progress Indicator* displays the number of equations which have converged to steady-state. The transient *Progress Indicator* displays the elapsed time of the transient. For example, the Converged Equations *Progress Indicator* shows that all 15 of the system equations have converged, and that, as shown by the Elapsed Time *Progress Indicator*, the transient simulation has progressed to 0.65 of a 2.50 seconds long transient.

The simulation is started by pressing the *Run Simulation* button located in the *System Control Dialog* window. The system first attempts to determine the steady-state (balanced) condition at the initial operating point, as was defined by the user. Once the engine is balanced, the transient will begin and run until the simulation time exceeds the Simulation Duration value entered in the *System Control Dialog*.

Graphing windows

Pressing the Graphing selection button on the *Main* window displays the *Graph Control Dialog* (see Figure 6). From this dialog, the user may select to graph a number of specified parameters for any of the components currently displayed in the *Engine Schematic Layout* window.

The *Graph Control Dialog* is comprised of two List components. The left-hand List displays each of the engine components currently displayed in the *Engine Schematic Layout* window. The right-hand List displays parameters to be graphed for the selected engine component. This List allows multiple selections, allowing the user to select to graph any or all of the component's parameters. To illustrate its use, consider the following example as depicted in Figures 6 and 7. Here the user has selected the component named MixVol 1. This component is an AeroMixingVolume object and as such has graph parameters of Volume, Stored Mass, Temperature, etc., which are shown in the right-hand List. From that List, the user has selected to graph the Temperature and Delta Temperature parameters during the transient. Figure 7 shows the temperature and pressure graphing windows which are displayed during the transient. Note that the user has also selected to graph the temperature and pressure parameters of the MixVol 3, MixVol 4 and MixVol 41 components for comparison.

Transcript window

The user may also view any non-critical text-based messages and status reports from the *Transcript* window (see Figure 8) during the simulation. The *Transcript* window may be displayed from the *Main* window.

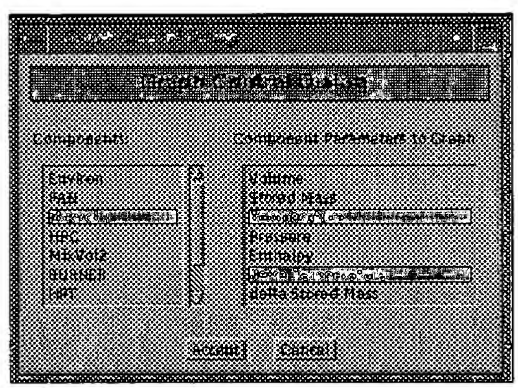

Figure 6: *Graph Control Dialog* window

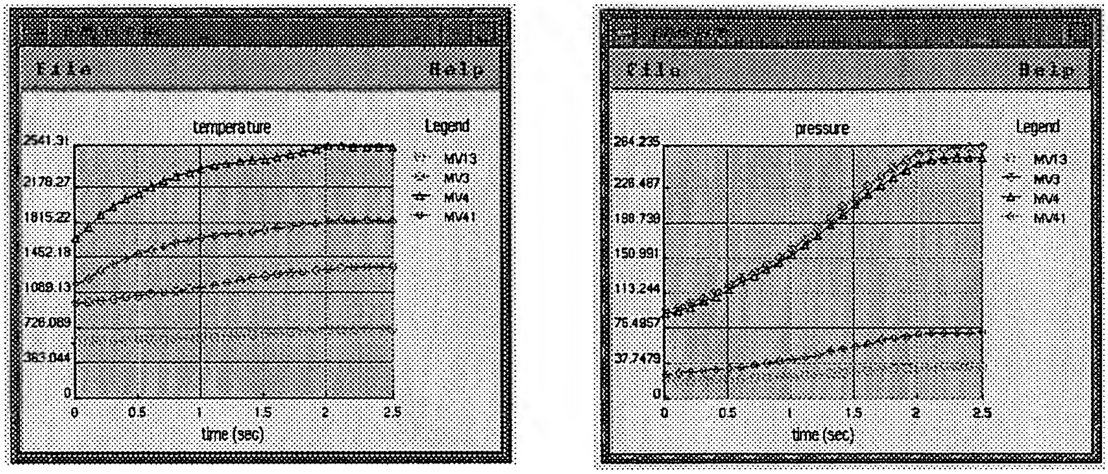

Figure 7: *Transient Graph* windows

Summary and Future Work

The Java Gas Turbine Simulation software reflects our initial effort in evaluating Java as a programming language and environment for developing a computing framework for aircraft simulation. Java provides a number of advantages in comparison to other object oriented programming languages, including the use of byte-code formats and a virtual machine architecture for increased portability, automated memory management, and a platform independent windowing system. By using Java, the simulation framework is well poised to take advantage of the rapid growth in networked computers. The Java run-time system, through its built-in support of network communication tools such as Sockets, Remote Method Invocation (RMI), and access to the WWW, as well as its capability to run on multiple platforms in heterogeneous networks provides an effective multi-computer approach for computationally intensive tasks.

Future work on the Java Gas Turbine Simulation will take advantage of enhancements and additions to Java which is expected with the release of version 1.1 of the Java Development Kit (JDK) in early 1997. These include the use of Remote Method Invocation (RMI) to invoke and distribute objects across a set of networked computers, and the development of a database interface using the Java Database Application Interface (JDBC).

Finally, previous work in AVS in which an expert system and visualization system were included to provide capability for intelligent monitoring and control of more advanced simulations conditions such as those experienced in engine component zooming, will be incorporated into the Java Gas Turbine Simulator. [7, 15].

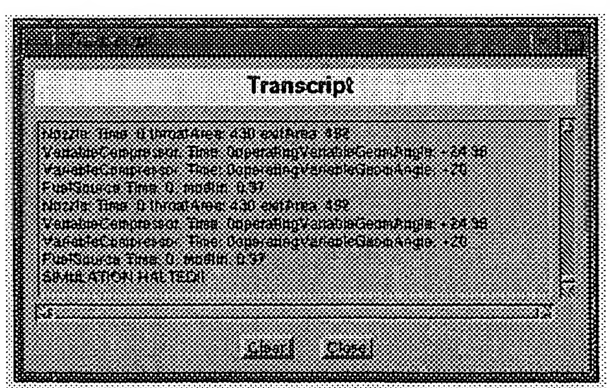

Figure 8: *Transcript* window

Acknowledgments

We would like to acknowledge the NASA Lewis Research Center (LeRC) Computing and Interdisciplinary System Office (CISO) for partial support of this work (Grant No. NCC-3-207). In particular, we would like to thank Greg Follen for his continued support. J. A. Reed is partially supported by a University of Toledo Doctoral Fellowship. For more information on the Java Gas Turbine Simulator software, send e-mail to jreed@top.eng.utoledo.edu.

References

[1] Claus, R. W., Evans, A. L., Lytle, J. K., and Nichols, L. D. "Numerical Propulsion System Simulation," *Computing Systems in Engineering,* 2, 4 (Apr. 1991), 357-364.

[2] Reed, J. A. and Afjeh, A. A., "An interactive graphical simulation environment for the study of gas turbine operation," AIAA paper 96-2563, July 1996.

[3] Advanced Visual Systems Inc., "AVS Developer's Guide (Release 4.0)," Part number: 320-0013-02, Rev B, Advanced Visual Systems Inc., Waltham, MA, May 1992.

[4] Reed, J. A. and Afjeh, A. A., "Distributed and parallel programming in support of zooming in numerical propulsion system simulation," *OAI/OSC/NASA Symposium on Application of Parallel and Distributed Computing*, Columbus, Ohio, April 1994, 176-184.

[5] Reed, J. A. and Afjeh, A. A., "Development of an interactive graphical propulsion system simulator," AIAA paper 94-3216, June 1994.

[6] Reed, J. A. and Afjeh, A. A., "An interactive graphical system for engine component zooming in a numerical propulsion system simulation," AIAA paper 95-0118, January 1995.

[7] Afjeh, A. A., Homer, P. T., Lewandowski, H. S, Reed, J. A., and Schlichting, R. D., "Development of an intelligent monitoring and control system for a heterogeneous numerical propulsion system simulation," *Proc. 28th Annual Simulation Symposium*," Phoenix AZ, April 1995, 278-287.

[8] Reed, J. A., "Development of an interactive graphical propulsion system simulator," Master of Science Thesis, The University of Toledo, Toledo, Ohio, August 1993.

[9] Booch, G, "Object Oriented Design with Applications," The Benjamin/Cummings Publishing Company, Inc., New York, 1991.

[10] Gosling, J., Joy, B., and Steele, G., "The Java Programming Language (version 1.0)," Addison Wesley Longman, July 1996.

[11] Reed, J. A. and Afjeh, A. A., "A Java Simulator for Teaching Gas Turbine Operation," AIAA paper 97-0850, January 1997.

[12] Daniele, C. J., Krosel, S. M., Szuch, J. R., and Westerkamp, E. J., "Digital Computer Program for Generating Dynamic Engine Models (DIGTEM)," NASA TM-83446, 1983.

[13] Holt, G., and Phillips, R., "Programming in NPSS, Phase II Report," NASA CR-NAS3-25951, November 1991.

[14] Goldberg, Adele, and Robson, David, "Smalltalk-80: The Language and Its Implementation," Addison-Wesley, 1985.

[15] Java Expert System Shell Manual. Ernest Friedman-Hill, Sandia National Laboratories. JESS Version 2.0, May 6, 1996.